A NEW TECHNIQUE FOR TEXT DATA COMPRESSION

UDITA KATUGAMPOLA
SOUTHERN ILLINOIS UNIVERSITY, CARBONDALE, IL 62901

Abstract. In this paper we use ternary representation of numbers for compressing text data. We use a binary map for ternary digits and introduce a way to use the binary 11-pair, which has never been use for coding data before, and we futher use 4-Digits ternary representation of alphabet with lowercase and uppercase with some extra symbols that are most commonly used in day to day life. We find a way to minimize the length of the bits string, which is only possible in ternary representation thus drastically reducing the length of the code. We also find some connection between this technique of coding data and Fibonacci numbers.

Keywords: Coding, Data Compression, Binary system, Ternary System, Golden Ratio

Introduction

Ternary or trinary is the base-3 numeral system. A ternary digit, *trit* contains about 1.58596 ($log_2 3$) bit of information. Even though ternary most often refers to a system in which the three digits, 0, 1, and 2, are all nonnegative integers, the adjective also lends its name to the balanced ternary system, which uses -1, 0 and +1 instead, used in comparison logic and ternary converters [1][2][3][4].

Techniques have been developed for text document compression that are semiadaptive, which uses frequncy-ordered array of word-number mappings [5][6][7]. Encoding binary digital data in ternary form has applicability to digital data communication systems and magnetic data storage systems [8] and also in high-speed binary multipliers, which uses ternary representations of two numbers [9]. Methods have been developed for converting binary signals into shorter balanced ternary code signals [2].

Variable length coding is a widely-used method in data compression, especially, in the applications of video data communication and storage, for example, JPEG, MPEG, CCITT H.261 and so on. Most of those methods implement the coding with two-field codes. Hsieh [10] introduced a method using three-field representation for each code. Also, we can adopt ternary systems in difference coding in audio compression[11][12]. The node of a Peano Curve can be represented by a base-3 reflected Gray Codes, RGC[13].

Ternary systems have been used in digital data recording systems [14] and precoded ternary data transmission systems as well [15] from which they can send and store more data compared to what binary system does.

Digital data compression is an important tool because it can be utilized, for example, to reduce the storage requirements for files, to increase the rate at which data can be transferred over bandwidth limited communication channels, and to reduce the internal redundancy of data prior to its encryption in order to provide increased security.

In this paper we use standard ternary representation for coding data. We introduce a new representation called $Base\, B_{23}$, which uses both ternary and binary representation with the maximum use of both binary and ternary features in a single coded data. Before going further, let us compare the binary and ternary representaions of few numbers,

$$85_{10} = 1010101_2 = 10010_3 \text{ and } 150_{10} = 10010110_2 = 12120_3$$

Thus, the binary represenatation of 85 uses 7 bits while ternary representation uses only 5 trits, and in 150, it uses 5 trits again, which is more than 40% saving in memory. Here we also notice that the 12-pair occurs twice in this representation. Thus if we can use a better representation which can code the 12-pair we can save more memory when saving these data and can save more time when sending to another detector. Thus our main goal is to develpoe a system which uses ternary representation in a sophisticated mannar. That is what we are going to do in the rest of this paper.

The Basic Definitions

We begin with the following two definitions.

Definition 1. Let $n = \epsilon_1\epsilon_2\epsilon_3...\epsilon_k$, where ϵ_i =0,1, 2; $k \in N$ be the ternary representation of the decimal number n. Then we say n is in the form A_{23}, if there is a map $\varphi : \{0, 1, 2\} \longmapsto \{00, 01, 10\}$ such that,

$$\phi(\epsilon_i) = \begin{cases} 00, & \text{if } \epsilon_i = 0; \\ 01, & \text{if } \epsilon_i = 1; \\ 10, & \text{otherwise.} \end{cases}$$

for each $i = 1, 2, 3, ..., k$.

With this definition, we can write $85_{10} = 10010_3 = 0100000100_{A_{23}}$ and $150_{10} = 12120_3 = 0110011000_{A_{23}}$. Thus it double the lenth of the string which represents the number in A_{23} base. But in this format we waste the 11 pair. So we need to modify the coding so that we can make use of the binary string 11_2. Thus we come up with the following definition, with the so called *Base* B_{23}.

Definition 2. Let $n = \epsilon_1\epsilon_2\epsilon_3...\epsilon_k$, where ϵ_i =0,1, 2; $k \in N$ be the ternary representation of the decimal number n. Then we say $n = \varepsilon_1\varepsilon_2\varepsilon_3...\varepsilon_l$, where $\varepsilon_i = 00, 01, 10, 11$; $l \leq k$, is in Base B_{23}, if there is a map $\psi : \{0, 1, 2\} \longmapsto \{00, 01, 10, 11\}$ such that $\psi(\epsilon_i) = \varepsilon_j$,where

$$\varepsilon_j = \begin{cases} 00, & \text{if } \epsilon_i = 0; \\ 01, & \text{if } \epsilon_i = 1 \, and \, \epsilon_{i+1} \neq 2; \\ 10, & \text{if } \epsilon_i = 2 \, and \, \epsilon_{i-1} \neq 1; \\ 11, & \text{if } \epsilon_i\epsilon_{i+1} = 12. \end{cases}$$

for each $i = 1, 2, 3, ..., k$.

That is, we say a number n is in base B_{23}, if each of the trits of its ternary representation is replaced by the following binary bits in that order:

$$0 \mapsto 00,$$
$$12 \mapsto 11,$$
$$1 \mapsto 01,$$
$$2 \mapsto 10.$$

So we can write $85_{10} = 0100000100_{A_{23}} = 0100000100_{B_{23}}$ and $150_{10} = 0110011000_{A_{23}} = 111100_{B_{23}}$. Thus it drastically reduces the length of the bits string if the 12_3 is present in the coding. The more 12_3 pairs are present, the more compact the code can be. Therefore, it is natural to seek the availability of 12_3 pairs in a string of ternary representation of a data. Thus we have the following lemma,

Lemma. *Golden Lemma*

The number S_n of ways that a string of trits 0, 1 and 2 of length n with no 12 pairs in the string is

$$S_n = \frac{1}{\sqrt{5}}\left\{\phi^{2n+2} - \phi^{-2n-2}\right\} \text{ for } n \in \mathbf{N},$$

where ϕ is the *Golden Ratio* $\frac{1+\sqrt{5}}{2}$.

Proof. We prove this result using a recurrence relation. First consider the following construction of string of 0,1 and 2 of size n.

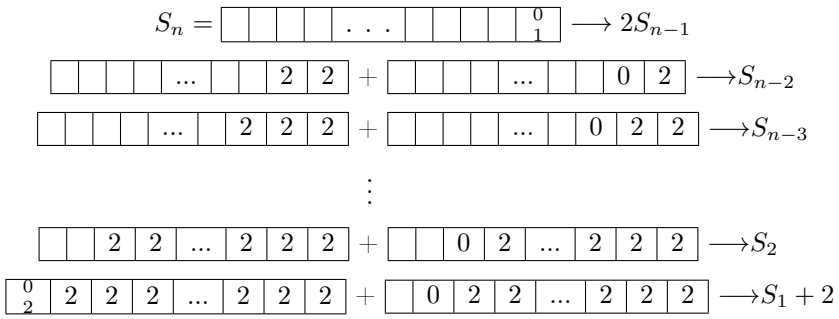

Figure 1: Deriving Formula

According to the diagram we have,

$$S_n = 2S_{n-1} + S_{n-2} + S_{n-3} + ... + S_2 + S_1 + 2,$$

with $S_1 = 3$ and $S_2 = 8$ for $n \geq 3$. This reduces to $S_n = 3S_{n-1} - S_{n-2}$ with $S_1 = 3$ and $S_2 = 8$ for $n \geq 3$. Solving the recurrence relation with the given data, we have,

$$S_n = \frac{1}{\sqrt{5}} \left\{ \left(\frac{1+\sqrt{5}}{2} \right)^{2n+2} - \left(\frac{1-\sqrt{5}}{2} \right)^{2n+2} \right\} \text{ for } n \geq 3$$

$$= \frac{1}{\sqrt{5}} \left\{ \phi^{2n+2} - \phi^{-2n-2} \right\}, \text{ where } \phi = \frac{1+\sqrt{5}}{2}.$$

□

The first few terms of this sequence are $S_1 = 3$, $S_2 = 8$, $S_3 = 21$, $S_4 = 55$, $S_5 = 144$. These are the even terms of the Fibonachchi sequence defined by, $F_n = F_{n-1} + F_{n-2}$ with $F_1 = 2$, $F_2 = 3$.

Now, consider a uniformely distributed ternary string of 0, 1, and 2 of size n. Then the following holds:

Theorem. *The probability of appearing at least one* $12-$ *pair of a string of* 0, *1, and* 2 *of length* n *is asymptotically 1.*

Proof. If S_n is the number of ways that a string of trits 0, 1 and 2 of size n can be arranged with no $12-$pairs, then the probability of getting at least one pair of 12 is

$$P_n = \frac{3^n - S_n}{3^n},$$
$$= 1 - \frac{1}{3^n \sqrt{5}} \left(\phi^{2n+2} - \phi^{-2n-2} \right)$$

Therefore, $\lim_{n\to\infty} P_n = \lim_{n\to\infty} \left(\frac{3^n - \frac{1}{\sqrt{5}} \phi^{2n+2}}{3^n} \right) = \lim_{n\to\infty} 1 - \frac{\phi^2}{\sqrt{5}} \left(\frac{\phi^2}{3} \right)^n \to 1$, since $\phi^2 = \frac{3+\sqrt{5}}{2} < 3$. □

This shows that when n gets larger, the mapping $12 \to 11$, should efficiently reduce the length of the B_{23} string.

Data compression with B_{23}

Here we discuss one of the main usage of B_{23} in coding data in day to day life. When compared to other benifits of converting data into binary form, word processing takes the leadership. Thus in this paper we discuss a technique that could drastically increase the storing and sending capabilities of data using the so called B_{23}ternary coding.

Algorithm. *Let* $A = \{a_1, a_2, a_3, ..., a_n\}$ *be an alphabet with,* $p(a_i)$ *being the probability of* a_i *appearing in the language generated by A. Without loss of generality, suppose* $p(a_1) \geq p(a_2) \geq ... \geq p(a_n)$. *Let* $B = \{b_1, b_2, b_3, ..., b_n\}$ *be a sequence of integers written in ternary form. Define,* $\delta : B \mapsto \mathbb{Z}$ *by*

$$\delta(b_i) = j, \text{ where } b_i = ...12_1...12_2...12_j...$$

Suppose without loss of generality, $\delta(b_1) \geq \delta(b_2) \geq ... \geq \delta(b_n)$. *Then we define a map* $\Omega : A \longmapsto B$ *such that* $\Omega(a_i) = b_i$ *for i=1,2,...,n.*

We notice that this scheme genarates a coding for the language generated by A. Now, we turn to a more practical example. That is the english alphabet and the language generated by it.

Example. Here we are going to find a B_{23} code for 26-letter english alphabet, which is different from Huffman coding of 26-letter alphabet and also is different from the ternary Huffman coding[11][16][17][18]. We also include some extra characters, which are frequently used in word processing. Before going further, we need to observe some facts related to the usage of english alphabet. When we use a language, we have to use some letters more frequently than others. Particularly in English the letter $'e'$ has much highest frequency compared to the others characters in the alphabet, while the letter $'z'$ has the lowest frequency. Some of these facts is summaries in the follwing table accompanied with the two charts 1 and 2. [1]

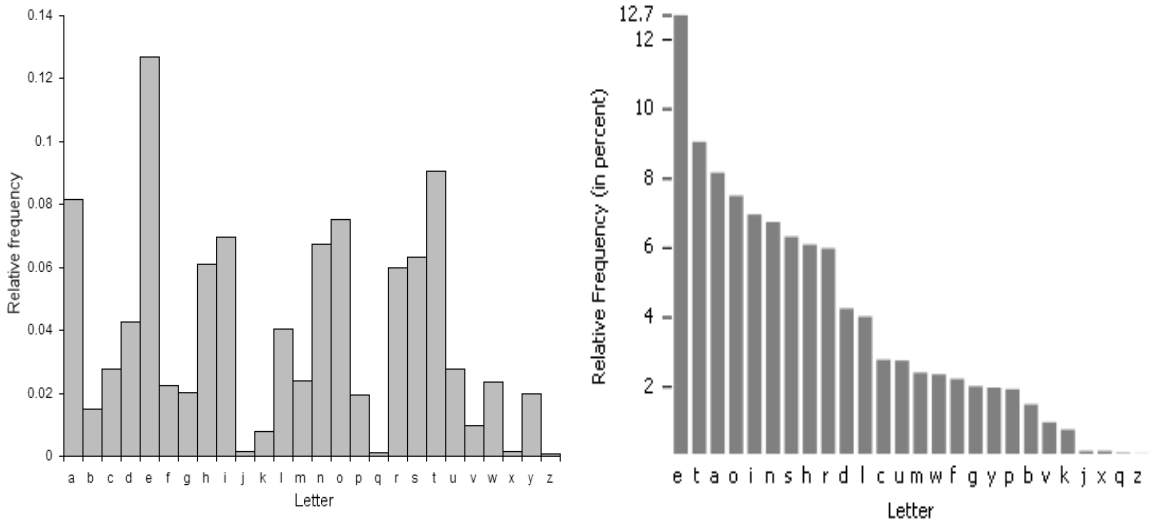

Figure 2: Bar-chart of frequency order of 26-letter English alphabet

According to the table below, the letters, e, t, a, o, i, n, s, h, and r has comparatively high frequency than the characters p, b, v, k, j, x, q, and z.Therefore if we can use a shoter code for the high frequence letters and comparatively longer code for low frequency characters we can possible reduce the string lenth for the coding sequence. This is the whole goal in the rest of the paper.

Letter	Frequency(%)	Letter	Frequency(%)
a	8.167	n	6.749
b	1.492	o	7.507
c	2.782	p	1.929
d	4.253	q	0.095
e	12.702	r	5.987
f	2.228	s	6.327
g	2.015	t	9.056
h	6.094	u	2.758
i	6.966	v	0.978
j	0.153	w	2.360
k	0.772	x	0.150
l	4.025	y	1.974
m	2.406	z	0.074

Table 1: Frequency order of 26-letter English alphabet

[1]Cryptographical Mathematics, Robert Edward Lewand. MAA, Washington DC, 2000

Above table force us to reorder the alphabet so that we use a short code for high frequency letter, while longer code for low frequncy letters. Before doing this we have to notice few things that are characteristics to the English alphabet. One major thing is alphabet has only 26 characters. Thus if we use a ternary coding with three trits, we can code all the 26 characters, but if we use binary coding we have to use 5 bits to handle these characters. This is the major observation in ternary system compared to binary system.

3 Exchanged Character Map - 3ECM

To represent high frequency characters with short code, we simply exchanged the positions of the English alphabet. Letters *e*, *t*, *a*, *o*, ... get shoter codes while the letters *v*, *k*, *x*, *q*, and *z* get comparatively longer codes as in the Huffman-encoding. What remains is to consider which upper case letters, typically the first letter of a word, appear more frequently as the first letter of a word. The top ten letters with frequencies, which occur at the beginning of words are:

Letter	T	A	I	S	O	C	M	F	P	W
Frequency(%)	15.94	15.5	8.23	7.75	7.12	5.97	4.26	4.08	4.0	3.82

Table 2: Frequency of the first letters

Clearly, the order differs from that for lower case (cf. t & e Vs T & E). Thus we propose the following character map with few extra sysmbols and accompanied ternary codes:

Dec	Symbol	Ternary	Dec	Symbol	Ternary	Dec	Symbol	Ternary
0	W	0000	27	z	1000	54	!	2000
1	N	0001	28	p	1001	55	$	2001
2	B	0002	29	b	1002	56	^	2002
3	C	0010	30	w	1010	57	%	2010
4	D	0011	31	x	1011	58	√	2011
5	T	0012	32	e	1012	59	,	2012
6	F	0020	33	f	1020	60	*	2020
7	G	0021	34	g	1021	61	/	2021
8	H	0022	35	v	1022	62	=	2022
9	P	0100	36	q	1100	63	<	2100
10	J	0101	37	j	1101	64	>	2101
11	K	0102	38	k	1102	65	@	2102
12	L	0110	39	y	1110	66	&	2110
13	M	0111	40	m	1111	67	’	2111
14	A	0112	41	n	1112	68	“	2112
15	O	0120	42	o	1120	69	?	2120
16	I	0121	43	a	1121	70	(	2121
17	S	0122	44	i	1122	71	)	2122
18	R	0200	45	r	1200	72	{	2200
19	Q	0201	46	s	1201	73	}	2201
20	T	0202	47	t	1202	74	⌊	2202
21	U	0210	48	u	1210	75	⌋	2210
22	V	0211	49	h	1211	76	\	2211
23	.	0212	50	Space	1212	77	;	2212
24	X	0220	51	d	1220	78	:	2220
25	Y	0221	52	l	1221	79	+	2221
26	Z	0222	53	c	1222	80	-	2222

Table 3: Coding Table

In our table the ternary code of almost all most frequent letters contain at least one 12 pair. Applying the B_{23} scheme results in the following code table.

According to the table, what we achieve here is that high frequent characters has shorter lenth compared to the others. Let us exemplify the method. For that we use a test string and code it in two ways, using B_{23} and the standed $ASCII$ code and then compare the two bitstrings generated from these techniques. It is done in the following way. The first Algorithm converts the text directly into $B_{23} - code$, and Algorithms 2 and 3 convert it back into human readable characters.

Symbol	Ternary	B_{23}	Symbol	Ternary	B_{23}	Symbol	Ternary	B_{23}
W	0000	00000000	z	1000	01000000	!	2000	10000000
N	0001	00000001	p	1001	01000001	$	2001	10000001
B	0002	00000002	b	1002	01000010	^	2002	10000010
C	0010	00000100	w	1010	01000100	%	2010	10000100
D	0011	00000101	x	1011	01000101	√	2011	10000101
T	**0012**	000011	e	**1012**	010011	,	**2012**	100011
F	0020	00001000	f	1020	01001000	*	2020	10001000
G	0021	00001001	g	1021	01001001	/	2021	10001001
H	0022	00001010	v	1022	01001010	=	2022	10001010
P	0100	00010000	q	1100	01010000	<	2100	10010000
J	0101	00010001	j	1101	01010001	>	2101	10010001
K	0102	00010010	k	1102	01010010	@	2102	10010010
L	0110	00010100	y	1110	01010100	&	2110	10010100
M	0111	00010101	m	1111	01010101	’	2111	10010101
A	**0112**	000111	n	**1112**	010111	“	**2112**	100111
O	**0120**	001100	o	**1120**	011100	?	**2120**	101100
I	**0121**	001101	a	**1121**	011101	(	**2121**	101101
S	**0122**	001110	i	**1122**	011110	)	**2122**	101110
R	0200	00100000	r	**1200**	110000	{	2200	10100000
Q	0201	00100001	s	**1201**	110001	}	2201	10100001
T	0202	00100010	t	**1202**	110010	[	2202	10100010
U	0210	00100100	u	**1210**	110100	]	2210	10100100
V	0211	00100101	h	**1211**	110101	\	2211	10100101
.	**0212**	001011	Space	**1212**	1111	;	**2212**	101011
X	0220	00101000	d	**1220**	111000	:	2220	10101000
Y	0221	00101001	l	**1221**	111001	+	2221	10101001
Z	0222	00101010	c	**1222**	111010	-	2222	10101010

Table 4: Complete Coding Table

Algorithm 1. *Coding into B_{23} form. Here we assume that the text string only consists of the characters listed in the above table.*
Let u_{ij}:= {(W, 0000), (N, 0001), (B, 0002), ... ,(-, 2222)}, i = 1, 2, ..., 81; j = 1,2
Input: Text String S. Let l=length(S); NewString =[],
for i=0 to l do
if $u_{i1} = characterAt(S, i)$
NewString = NewString +u_{i2},
end do.

Once we received the string to the destination we use the following two algorithm to decode it. First one to transform it back into ternary and then the second one to decode it to human readable code.

Algorithm 2. *Converting into ternary.*
Input received string S. Let l=length(S/2), NewString = [].
For i=1 to l
if SubString(S,i,i+1) = ’00’ then NewString = NewString + ’0’,
elseif SubString(S,i,i+1) = ’01’ then NewString = NewString + ’1’

elseif SubString(S,i,i+1) = '10' then NewString = NewString + '2'
else NewString = NewString + '12'
i = i+2;
end do

Now we read this string as a four character groups and assign each such group into a single character according to the above table. So we use the following algoritm

Algorithm 3. *Decoding ternary into human readable characters*
Let u_{ij}:= {(W, 0000), (N, 0001), (B, 0002), ... ,(-, 2222)}, i = 1, 2, ..., 81; j = 1,2
Input: Ternary String S. Let l=length(S); NewString =[],
for i=0 to l do
if $u_{i2} = subString(S, i, i + 3)$
NewString = NewString +u_{i1},
else NewString = NewString+[],
i=i+4;
end do.

Upperbound for Compression Ratio. To derive an upperbound for the compression ratio, we have to make few assumptions since this compression is dynamic. Let us assume for a longer text at least the *space* has the highest frequency. In order to get a numerical value we assume the chance of *space* is 50% and the other letters a_i has the probabilities, p_i half of the table values. Then

$$\begin{aligned} Compression\ Ratio &= \frac{\text{Length of compressed text}}{\text{Length of uncompressed text}}, \\ &= \frac{\sum_{i=1}^{n} \text{length of letter } a_i \text{ after compression} \times p_i}{\sum_{i=1}^{n} \text{length of letter } a_i \text{ before compression} \times p_i}, \\ &\leq \frac{4 \times 100 + 6 * (8.167 + ... + 2.758) + 8 * (1.492 + ... + 0.074)}{8 \times 200}, \\ &= 0.64577 \end{aligned}$$

We can achieve much stronger compressions when we applied this scheme with the Huffman encoding. We can directly use the B_{23} scheme, once the technology develops to a level where we can use ternary bitstrings for data transmission.

Now we use the above algorithms to the following example, and cpmpare the results with the familier ASCII encoder. We notice that even for a short code we see noticable reduction of the coded string.

Example. So, consider the text string,

S = " This is the test message."

Once we apply the code into B_{23},we get,

$$\begin{aligned} CodedString = {} & 000011110101011110110001111101111011000111111100101101010100111111110010 \\ & 0100111100011100101111010101010100111100011100010111010100100101001100101 1 \end{aligned}$$

This can be compared with the corresponding Binary string generated by ASCII coding, which is 25% larger than the B_{23},

$$\begin{aligned} BinaryString = {} & 101010001101000011010010111001100100000011010010111001100100000011101000 1 \\ & 1010000110010100100000011101000110010101110011011101000010000001101101011 \\ & 0010101110011011100110110000101100111011001010010111 0 \end{aligned}$$

After using second algorithm, we end up getting,

DecodedString = "This is the test message."

Thus if we can adopt this technique in word processing and data compression we can drastically reduce the memory needed to store information and also in data transmission.

We can extend this technique for six-digits ternary system with more characters than in this case. That would be the nest task. We can also extend this technique with slight modifications for compressing highly randomized data [20]. We conclude the paper with the following question,

Question: What kind of distributions has more $12-$pairs in a string of $0, 1$, and 2?

Acknowledgement. Author would like to express his heart felt gratitude to Dr. Jerzy Kocik of Department of Mathematics of Southern Illinois University, for his invaluable suggestions and continued support.

References

[1] Gundersen, Henning B., Yngvar, Fast Addition Using Balanced Ternary Counters Designed with CMOS Semi-Floating Gate Devices, University of Oslo, Norway, May 2007, ISSN: 0195-623X
[2] Van et al, US Patent 4003041. System for converting binary signals into shorter balanced ternary code signals. filed Dec 30, 1974
[3] Van et al., US Patent 3599205. Binary to ternary protected code converter. filed Aug 28, 1968
[4] Gilbert et al., US Patent 3866147. Balanced correlated ternary coding system. filed Feb 26, 1973
[5] Kaplan et al., US Patent 5325091. Text-compression technique using frequency-ordered array of word-number mappers. June, 1994. 341/51
[6] Lisle et al., US Patent 4843389. Text compression and expansion method and apparatus. June, 1989
[7] Crandall, G.E. US Patent 5999949. Text file compression system utilizing word terminators. December, 1999
[8] Jacoby et al., US Patent 4506252. Ternary data encoding system. filed July 05, 1983.
[9] Fensch, T., US Patent 4628472. Binary multiplier using ternary code. filed Nov 18, 1983
[10] Hsieh, C., US Patent 5648774. Variable length coding with three-field codes. filed May 08, 1995
[11] Data Compression, 4ed. David Salomon, Springer-Verlag, London, 2007.
[12] Gray, F. US Patent 2632056. Pulse Code Communication. March 17, 1953.
[13] Cole, A.J.(1985) A Note On Peano Polygons and Gray Codes, International Journal of Computer Mathematics, 18:3-13.
[14] Potter et al., US Patent 3226685. Digital recording systems utilizing ternary, n bit binary and other self-clocking forms. December, 1965. 360/40
[15] Howson, R.D. US Patent RE30182. Precoded ternary data transmission. December, 1979. 375/290
[16] David Salomon, Variable-length Codes for Data Compression, Springer-Verlag, London, 2007.
[17] David Salomon, A Concise Introduction to Data Compression, Springer-Verlag, London, 2008.
[18] C.B, John, G. C, Ian, H.W., Text Compression, Timothy, Prentice Hall, Englewood Cliffs, New Jersey, 1990
[19] Robert Edward Lewand, Cryptological Mathematics, The Mathematical Association of America, Washington DC, 2000.
[20] James, D.C. US Patent 5533051. Method for data compression. July 1996, 375/240